The Coming Revolution in Physics Education

William Flannery, Berkeley Science Books

This paper describes a high school or introductory university course in scientific programming that introduces the computer revolution into the physics curriculum at the beginning. In the first one-hour lecture, Euler's method is presented and used to compute a solution to the analytically unsolvable two-body problem.[1] In the remainder of the course a variety of physical systems are modeled and analyzed to demonstrate the remarkable power and wide range of applicability of the method.

Introduction

Modern math and physics began in the 17th century when Isaac Newton discovered the law of gravity and the laws of motion, and derived the differential equation that governs the motion of two bodies acted on by their mutual gravity alone, i.e., the two-body problem. Newton developed analytic calculus to analyze differential equations, and since that time analytic calculus has been the mathematics of physics and engineering. However, most differential equations are difficult or impossible to solve, and a university student's first course in differential equations is typically in his/her sophomore or junior year.[2]

Computers make it easy to compute approximate solutions to differential equations without solving the equations analytically, and for this reason they have revolutionized science and engineering. The basic method of computational calculus is known as Euler's method. Euler's method is simple, intuitively clear, and easy to learn: it can be taught to high school science students with no previous exposure to calculus in a one-hour lecture. The content of that lecture begins the next section.

The analysis of physical systems

This section describes the analyses of several physical systems just as they are covered in the course, start to finish, with nothing omitted. The analysis of each system has four parts:

- The physical laws are stated.
- The process model is derived.
- Euler's method is used to translate the model to computational equations, which are then translated to MATLAB.
- The MATLAB model of the process is run and the results graphed.

Central force motion

The physics of central force motion is taught in introductory physics classes. The physics is Newton's law of gravity $F_g = G \cdot m_1 \cdot m_2 / r^2$, and Newton's second law of motion $F = m_1 \cdot A$.

The mathematical model for central force motion is derived by substituting the formula for the force of gravity into the second law of motion and dividing by m_1 to obtain

$$A = G \cdot m_2 / r^2 . \tag{1}$$

A differential equation is an equation for a rate of change. A differential equation model of a process consists of a set of variables that define the state of the process, the state variables, and an equation for the rate of change of each state variable. Following the usual convention, the function that computes the rate of change of a variable f is named f'. Newton's model for a falling object that is acted on only by Earth's gravity consists of state variables for position r and velocity v and the rate equations

$$r'(t) = v(t), \tag{2}$$

$$v'(t) = -G \cdot m_{\text{Earth}} / r(t)^2. \tag{3}$$

First we calculate an approximate trajectory of a falling object by hand. The object's initial position $r(t_1)$ and velocity $v(t_1)$ are given. Euler's method translates each rate equation to a computational equation:

$$r(t_{i+1}) = r(t_i) + v(t_i) \cdot \Delta t, \tag{4}$$

$$v(t_{i+1}) = v(t_i) + [-G \cdot m_{\text{Earth}} / r(t_i)^2] \cdot \Delta t. \tag{5}$$

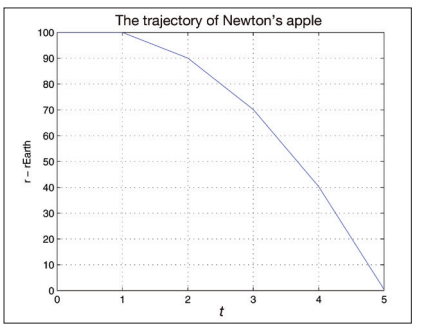


Fig. 1. The trajectory of Newton's apple.

Then, given the object's position $r(t_i)$ and velocity $v(t_i)$ at time t_i, we calculate position and velocity at time $t_{i+1} = t_i + \Delta t$ using the computational equations.

Figure 1 shows five seconds of an approximate trajectory for an apple dropped from a height of 100 m, calculated as follows, with $\Delta t = 1$ s.

$t_1 = 0$, $r(0) = 100 + r_{\text{Earth}} = 6370100$ and $v(0) = 0$

$t_2 = 1$

$r(1) = r(0) + v(0) \cdot 1 = 6370100$

$v(1) = v(0) + (-G \cdot m_{\text{Earth}} / r(0)^2) \cdot 1 = -9.9$

$t_3 = 2$

$r(2) = r(1) + v(1) \cdot 1 = 6370090$

$v(2) = v(1) + (-G \cdot m_{\text{Earth}} / r(1)^2) \cdot 1 = -19.9$

$t_4 = 3$

$r(3) = r(2) + v(2) \cdot 1 = 6370070$

$v(3) = v(2) + (-G \cdot m_{\text{Earth}} / r(2)^2) \cdot 1 = -28.9$

$t_5 = 4$

$r(4) = r(3) + v(3) \cdot 1 = 6370040$

$v(4) = v(3) + (-G \cdot m_{\text{Earth}} / r(3)^2) \cdot 1 = -39.8$

$t_6 = 5$

$r(5) = r(4) + v(4) \cdot 1 = 6370000$

$v(5) = v(4) + (-G \cdot m_{\text{Earth}} / r(4)^2) \cdot 1 = -49.7$

The graph is a plot of $r(t) - r_{\text{Earth}}$ vs. time. When Δt is large the solution is not accurate; as Δt becomes smaller the solu-

tion becomes more accurate. It can be shown that as Δt approaches 0 the approximate solution computed using Euler's method approaches the exact solution.[3] This marks the end of the material in the first lecture.

The computational equations above translate one for one to MATLAB statements. The complete program for the problem is

```
G = 6.7e-11; % Gravitational constant,N·m2/kg2
mEarth = 5.97e24;       % Mass of earth in kg
rEarth = 6.37e6;        % Radius of earth in m
N = 5;                  % Number of subintervals
dt = 1;                 % Length of a subinterval
r(1) = 100 + rEarth;    % Initial position
v(1) = 0;               % Initial velocity
t(1) = 0;               % Start time
for i = 1:N          % For-loop implementing Euler's
   t(i+1) = t(i) + dt;     % computational equations
   A = -G*mEarth / r(i)^2;      % Eq. (1)
   r(i+1) = r(i) + v(i)*dt;     % Eq. (4)
   v(i+1) = v(i) + A*dt;        % Eq. (5)
end
plot(t, r - rEarth)     % Plot the trajectory
xlabel('t')
ylabel('r - rEarth')
```

This program is the prototype for every program in the course, consisting of an initialization section, a for-loop to implement Euler's computational equations, and plot statements to graph the results.

With acceleration resolved into components, the x and y trajectories of a satellite are calculated just as the 1D trajectories in the previous example. This is the for-loop for computing trajectories in 2D:

```
for i=1:N
   t(i+1) = t(i) + dt;
   R = sqrt(x(i)^2+y(i)^2); % Distance to satellite
   A = - G*mEarth/R^2;     % Compute acceleration
   Ax = (x(i)/R)*A;     % Resolve acceleration into
   Ay = (y(i)/R)*A;        % x and y components
   x(i+1) = x(i) + vx(i)*dt; % Update x position
   vx(i+1) = vx(i) + Ax*dt;  % and velocity
   y(i+1) = y(i) + vy(i)*dt; % Update y position
   vy(i+1)= vy(i) + Ay*dt;   % and velocity
end             % See complete program at Ref. 4
```

Figure 2 shows three graphs produced by the program, with circular outlines of Earth added by the program for reference. The first is the space station orbit, the second an elliptical orbit, and the third a hyperbolic trajectory. The only change to the program was to vary the initial state (position and velocity) of the satellite and the simulation run time.

We can model a rocket launched from Earth toward the Moon by modeling both the rocket and the Moon. The rocket is pulled by Earth's gravity and the Moon's gravity, so the rocket acceleration is calculated using the sum of the gravitational forces of Earth and the Moon. We've gone from an analytically unsolvable two-body problem to an analytically intractable three-body problem (Earth, rocket, Moon). The code to update the rocket state variables xR, yR, vxR, and vyR is:

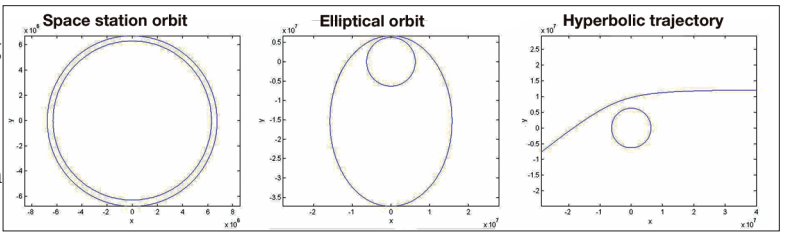


Fig. 2. Orbits. The smaller circle in each figure represents Earth.

```
xR(i+1) = xR(i) + vxR(i)*dt;  % Update rocket
yR(i+1) = yR(i) + vyR(i)*dt;  % position

RE = sqrt(xR(i)^2+yR(i)^2);  % Compute rocket
AE = - G*mEarth/RE^2;   % acceleration due
AEx = (xR(i)/RE)*AE;    % to earth and resolve
AEy = (yR(i)/RE)*AE;    % to x and y components

RM = sqrt((xR(i)-x(i))^2+(yR(i)-y(i))^2);
AM = - G*mMoon/RM^2;          % Compute rocket accel
AMx = (xR(i)- x(i))/RM)*AM;   % due to moon and
AMy = (yR(i)- y(i))/RM)*AM;   % resolve

vxR(i+1)= vxR(i) + (AEx + AMx)*dt; % Update
vyR(i+1)= vyR(i) + (AEy + AMy)*dt; % velocity
```

By shooting the rocket straight up and adjusting the rocket launch velocity and the initial Moon position, we can get the rocket to hit the Moon.

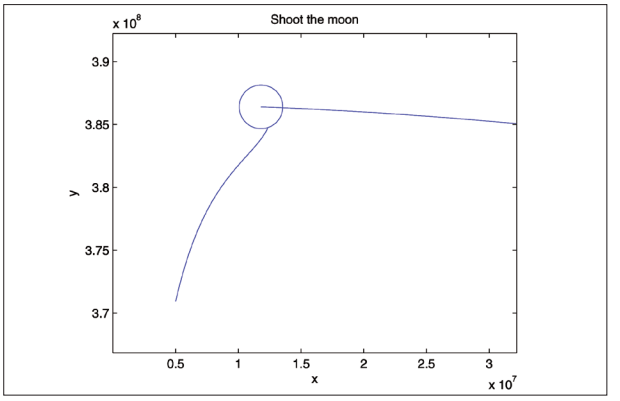


Fig. 3. Shoot the Moon.

A hit! The moon shot flight time is four days. Figure 3 shows the rocket and Moon trajectories for the last eight hours of the flight. The program adds the Moon's outline to the graph.

The rocket is being modeled as a point mass, and we are striving for simplicity, so we model a guidance boost on the rocket by adding the acceleration due to the boost to the rocket velocity as follows:

```
% Apply timed boost as rocket approaches moon
if (t(i) >= startBoost && t(i) < stopBoost) % Boost?
     vxR(i+1) = vxR(i+1) + xBoost*dt; % Yes
     vyR(i+1) = vyR(i+1) + yBoost*dt;
end
```

Now we can model a rocket that just misses the Moon and give it a slight nudge from a guidance boost as it passes by the Moon; the Moon's gravity does the rest, creating the Apollo trajectory, shown in Fig. 4.

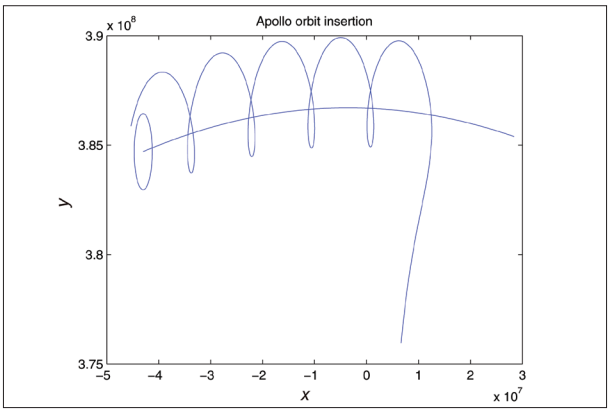


Fig. 4. Apollo orbit insertion.

Figure 5 shows the Apollo simulation with no guidance boost; the rocket flies by the Moon just missing it. As it flies by the Moon it is pulled by the Moon's gravity. Figure 6 shows the rocket's velocity profile as it flies by the Moon, with the top line in blue representing the rocket speed. The jump in speed is called the slingshot or gravity assist effect. Gravity assist trajectories play an essential role in rocket missions to the outer planets.

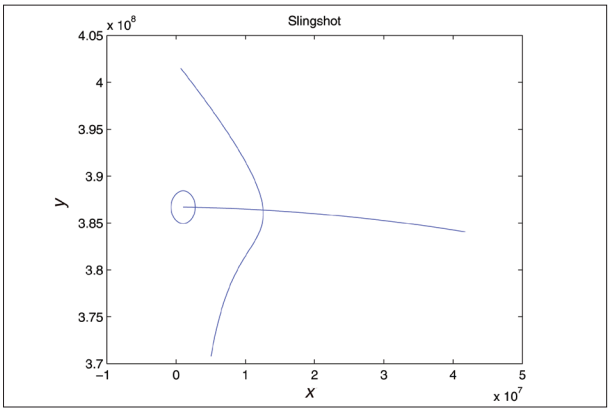


Fig. 5. Slingshot trajectory.

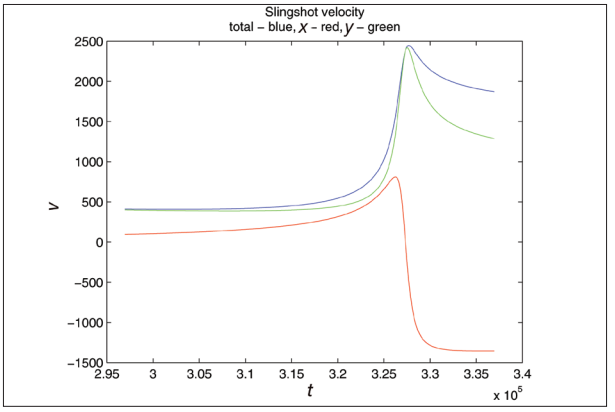


Fig. 6. Slingshot velocities.

On Aug. 5, 2011, the United States launched the Juno probe to Jupiter. The probe was launched from a low Earth orbit into a two-year orbit around the Sun. In October of 2013 the probe returned to Earth and the slingshot effect propelled the probe toward Jupiter; it arrived at Jupiter in July 2016 and will orbit Jupiter until 2021.

Figure 7 is a NASA graphic showing the Juno trajectory. After the launch the probe moves toward Jupiter under the influence of gravity only except for a Deep Space Maneuver, which we model as a single guidance boost.

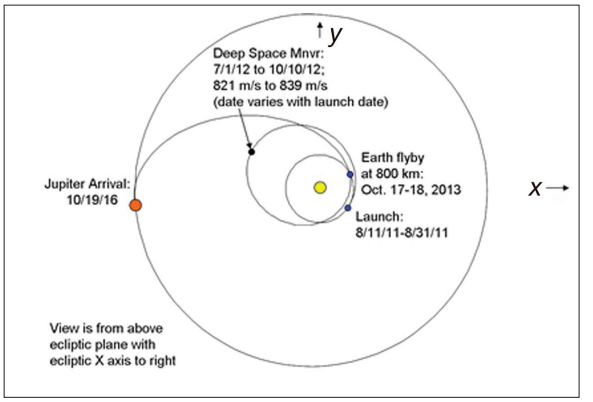


Fig. 7. NASA Juno trajectory.

The model equations for the Juno trajectory are identical to those for the Apollo mission, with the parameters for the rocket, Earth, and Sun for the Juno probe replacing the parameters for the rocket, Moon, and Earth of the Apollo program. Figure 8 shows the computed trajectories of the Juno probe and Earth for three years.

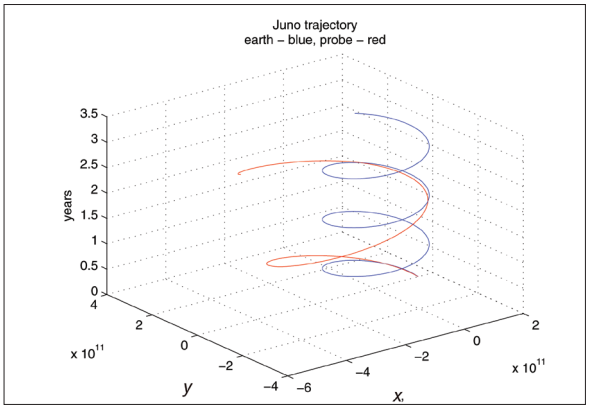


Fig. 8. The Juno trajectory.

The Voyager and Cassini missions also used slingshot/gravity assist trajectories. Charley Kohlhase was the mission planner for both projects, working for the Jet Propulsion Laboratory. Students can learn more about missions to the outer planets by watching his four-video series "Trajectory Design and Gravity Assist" online.[5]

Electric circuit analysis and 2D rigid body dynamics

The appendix contains analyses of problems in electric circuit analysis and 2D rigid body dynamics with the same level of detail as the analyses of the central force motion problems above.[6]

Programming notes

Writing a program completes the transformation of the abstract mathematical model of a process to a program model that can be tested, observed, and modified. Writing the program and testing it with various system parameters and initial conditions gives the student a sense of mastery of the physics, the process model, and the process itself.

Since programming is not a prerequisite for the course, the approach is to make the programming as simple as possible. In addition, the class should have a lab section, where students can work on the programs and get help to resolve any programming issues.

No algorithm development is required as the algorithm is the same for every program: a for-loop implementing Euler's computational equations. Also, each program has the same structure, consisting of three sections: initialization, a for-loop, and graphing,

The initialization section assigns values for physical constants, system parameters, initial conditions, and simulation parameters. These values can be discussed in class and provided in the assignment, possibly specifying a range of values for selected parameters and initial conditions.

Each process model is derived in class. Euler's method is used to translate the model into computational equations, and the values of the state variables for the first few time steps are calculated by hand, all in class. The translation from computational equations to MATLAB statements is one to one and by rote. At this point the student understands exactly what the program for-loop does, is able to write the for-loop code, and has a set of values to use to test the first few iterations of the programmed for-loop.

The graphing section is automated by MATLAB's plot functions and requires very little programming.

MATLAB is the industry standard programming language for engineering applications; the basic language hasn't changed in 30 years. We use only the basic features, variables and arrays, assignment, if, and for statements, plus one high-level feature, MATLAB's excellent graphing capabilities.

Python is another programming language that is popular in academic physics, and it would work equally well.

Improving physics education

The early introduction of differential equations, but not computational calculus, into the university engineering curriculum is one of the primary features of an ongoing NSF-sponsored project at Wright State University that has had great success.[7]

Computational calculus is one of the primary components of computational physics, and there is a growing awareness that universities have been slow to incorporate computational physics into the physics curriculum. A group of physics professors, Partnership for Including Computation in Undergraduate Physics (PICUP),[8] has formed to promote the incorporation of computational methods into university undergraduate physics education. The PICUP approach is 'top down', in that the goal is to introduce computational methods into already existing physics courses.[9,10] One well-known textbook integrates computational methods, but not differential equations, into introductory college level physics.[11]

The proposed course represents a new approach that is 'bottom up' and introduces computers, differential equations, and computational calculus into the physics curriculum at the beginning, independently of the math curriculum beyond high school algebra and geometry.

Summary

This introductory course in scientific programming produces results that are seen in the university in the upper division or beyond. How is this possible?

- Differential equations are used to model physical processes.
- Euler's method is used to compute solutions to the models.
- MATLAB programming is used to implement Euler's method.

These factors are interdependent and each one is crucial. The laws governing physical processes are written as differential equations, and physical processes are modeled with differential equations, so this is the essential starting point.

Even though the models are simple, they are difficult or impossible to solve analytically. Euler's method is simple and powerful; it translates the differential equations to arithmetic equations and takes the mystery out of differential equations.

Thousands of calculations are required to get accurate answers. Thus computer programming is necessary. The translation from Euler's computational equations to MATLAB assignment statements is one to one and by rote.

The course is self-contained and independent of the calculus curriculum, and can be offered as early as the junior year in high school. A pilot ungraded version of the course has been successfully taught to a small class of seniors at Shorecrest Prep School in St. Petersburg, FL.

Differential equations have extraordinary analytic and explanatory power, as we've seen in this short paper. But it takes computational calculus to unleash this power. Introducing students early to modeling with differential equations and analyzing a wide range of physical systems using computational calculus, from the Juno space probe now orbiting Jupiter to the VEX robot on the classroom floor, will put computers, differential equations, and computational calculus at the center of technical education from the beginning. In the future each will play an increasingly important role in many of the courses in the physics curriculum.

The course has additional benefits: it demonstrates the central role that differential equations play in physics, and it will demystify and motivate the study of analytic calculus. It provides the perfect lead up to the next challenge in mathematical physics: systems described by partial differential equations, which can also be taken up early using computational methods.[12]

Computers and computational calculus will transform physics and STEM education for this reason: they make it easy to analyze differential equation models. The analyses of complex physical systems that they make possible, and the fantastic results that can be obtained, will empower and motivate students and energize the entire physics curriculum.

References

1. F. Fitzpatrick, "Newtonian Dynamics." There is no closed-form function $r(t)$ such that $r'' = G \cdot m / r^2$. Newton derived Kepler's laws, from which Newton and Kepler derived numerical procedures to approximate $r(t)$. http://farside.ph.utexas.edu/teaching/336k/lectures/node32.html.
2. USF Physics program, http://physics.usf.edu/ug/program. A physics major takes calculus courses MAC 2311, 2312, and 2313

in the first two years, and a course in differential equations in the junior year, course PHZ 3113.
3. F. Fitzpatrick, "Integrating ODEs," http://farside.ph.utexas.edu/teaching/329/lectures/node35.html.
4. MATLAB programs at http://www.berkeleyscience.com/MMCCIprograms/.
5. C. Kohlhase, "From Ellipses to Gravity Assists," https://web.archive.org/web/20160412033718/http://saturn.jpl.nasa.gov/video/products/MultimediaProductsPresentationsSlides/.
6. View the appendix at *TPT Online*, http://dx.doi.org/…, under the Supplemental tab.
7. N. Klingbeil, R.Mercer, K. Rattan, M. Raymer, and D. Reynolds, The WSU Model for Engineering Mathematics Education, in *2005 ASEE Annual Conference and Exposition: The Changing Landscape of Engineering and Technology Education in a Global World, June 12, 2005 – June 15, 2005* (American Society for Engineering Education, 2005).
8. PICUP, https://www.compadre.org/PICUP/.
9. N. Chonacky and D. Winch, "Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments," *Am. J. Phys.* **76**, 327–333 (April 2008).
10. R. Chabay and B. Sherwood, "Computational physics in the introductory calculus-based course," *Am. J. Phys.* **76**, 307–313 (April 2008).
11. R. Chabay and B. Sherwood, *Matter & Interactions*, 4th ed. (Wiley, 2015).
12. W. Flannery, "Mathematical Modeling and Computational Calculus II," http://www.berkeleyscience.com/MMCCII.htm.
13. SPICE, Wikipedia, https://en.wikipedia.org/wiki/SPICE, freeware: http://www.analog.com/en/design-center/design-tools-and-calculators.html.

William Flannery *graduated from the University of California at Berkeley with an MS degree in mathematics and a PhD in Computer Science. He worked for 20 plus years as an avionics and control systems engineer at high tech firms, among them Honeywell Aerospace (first use of laser gyros in space), Lockheed (sensor integration on the YF-22), Stanford Telecom (GPS system simulator), and Berkeley Process Control (advanced process control). He has written two books on the use of computers in physics and engineering,* Mathematical Modeling and Computational Calculus, *Volumes I and II, and developed a video series for the books now available on YouTube.*
wdflannery@aol.com

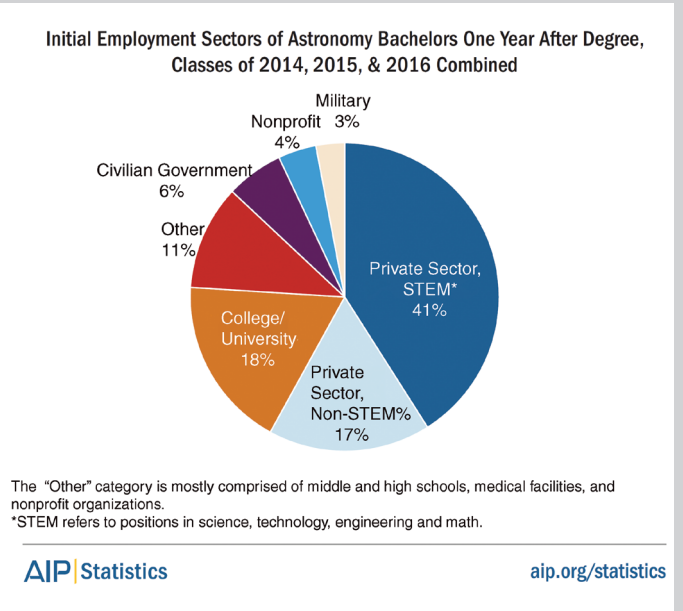
Appendix: next page

And the Survey Says…

***Susan C. White,* Column Editor**
American Institute of Physics
Statistical Research Center
College Park, MD 20740; swhite@aip.org

Where do astronomy bachelor's degree recipients work?

As we noted in last month's column, half of the astronomy bachelor's degree recipients were in the workforce one year after earning their degrees. Among those accepting employment, about one in eight (13%) were continuing in positions they held prior to receiving their degrees. About one in seven (14%) indicated that they planned to enroll in graduate school starting the next academic year, with an additional 12% enrolling in two or more years.



New astronomy bachelors secured employment in many sectors of the economy, with over half (58%) working in the private sector. More than two-thirds of those employed in the private sector indicated they were working in a STEM (science, technology, engineering and math) field. Most of these positions were in either engineering or computer science, with the remainder split between physics, astronomy, and other technical fields. Common job titles included "software developer" and "research analyst." Astronomy bachelors in private sector non-STEM positions worked in a variety of positions ranging from financial analysis to retail. About half of the bachelors in these non-STEM private sector positions indicated they were solving technical problems on a daily or weekly basis.

In November, we will look at the salaries some of these graduates earned.

Susan White works in the Statistical Research Center at the American Institute of Physics. She can be reached at swhite@aip.org.

The Coming Revolution in Physics Education, by William Flannery

Appendix

The analysis of physical systems (cont.)

Electric circuit analysis

The physics of electric circuit analysis is simple and intuitively clear, it consists of three component models and Kirchoff's laws:

- ***Transistor model:*** Ohm's law, $v_R = I_R \cdot R$, the resistor voltage v_R equals the current I_R through the resistor times the resistor's resistance R.
- ***Capacitor model:*** $v_C' = I_C / C$, the rate of voltage increase across a capacitor v_C' equals the current flow into the resistor I_C divided by the capacitor's capacitance C.
- ***Inductor model:*** $I_L' = v_L / L$, the rate of current increase in an inductor I_L' equals the applied voltage v_L divided by the inductor's inductance L.
- ***Kirchoff's loop law:*** The voltages across the components in a loop sum to 0, and Kirchoff's node law: the currents into a node sum to 0.

We begin by analyzing single loop circuits containing one or two components, to study the component characteristics. Then circuits with more components are analyzed, e.g. the RLC oscillator shown in Fig. A1.

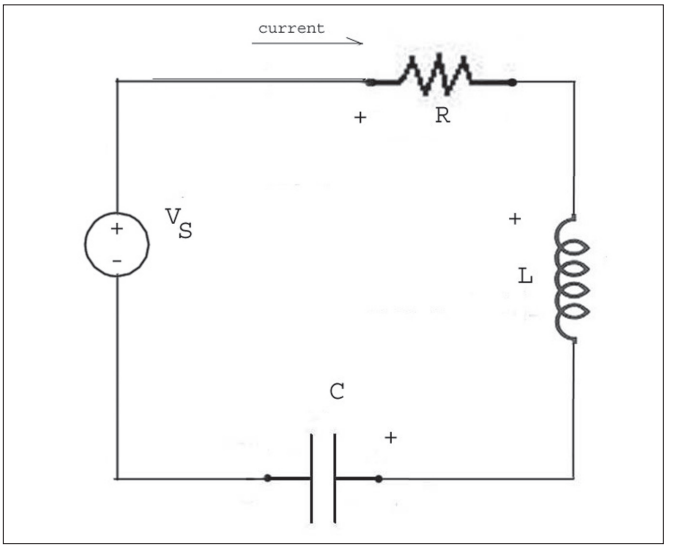


Fig. A1. RLC oscillator circuit.

There is a state variable for each component that stores energy, i.e. each capacitor and inductor; the state variables for the RLC circuit above are v_C and I_L.

From the capacitor model

$$v_C'(t) = I_C(t)/C = I(t)/C, \tag{1}$$

noting that in the single loop $I_C = I_R = I_L = I$.

From Kirchoff's loop law and the resistor model,

$$v_L(t) = V_S(t) - I_R(t)\cdot R - v_C(t). \tag{2}$$

and from the inductor model

$$I_L'(t) = v_L(t)/L = [V_S(t) - I_R(t)\cdot R - v_C(t))/L. \tag{3}$$

The for-loop code for the RLC circuit model is:

```
for i=1:1000
  t(i+1) = t(i) + dt;
  vc(i+1) = vc(i) + I(i)/C*dt;
  I(i+1)= I(i) + (Vs(i) - vc(i) - I(i)*R)/L*dt;
end          % See complete program at Ref. 4
```

V_S is a programmable voltage source. With $d_t = 0.01$ s, a unit impulse at $t = 1$ s is programmed as follows:

```
VS = zeros(1,1000);

VS(101)= 100;
```

V_S is graphed in Fig. A2 and the RLC circuit response is graphed in Fig. A3.

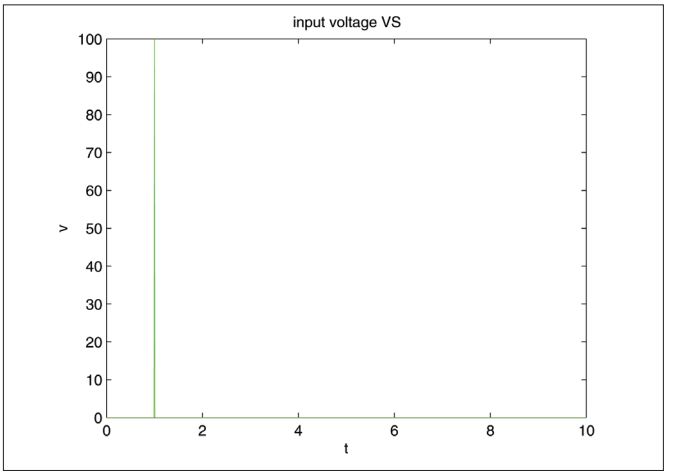


Fig. A2. Input voltage VS.

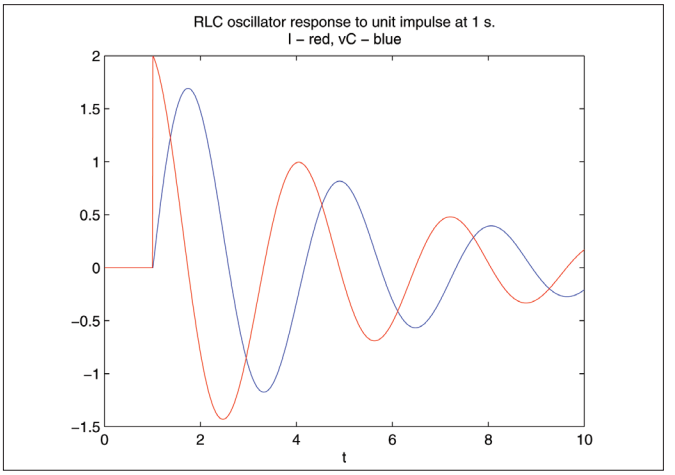


Fig. A3. RLC circuit response.

2D rigid body dynamics

The physics of 2D rigid body dynamics is:

- Newton's second law for translational motion: $F = m \cdot p''$ where p is the position of the object's center of mass.
- Newton's second law for rotation: $\Gamma = I \cdot \alpha''$ where Γ is the applied torque, I is the object's moment of inertia about its center of mass, and α is the object's orientation angle.

A 3-stage Delta IV rocket in Fig. A4 is modeled as a cylinder, and steered using a programmed thrust angle θ.

Fig. A4. Delta IV rocket and model.

The forces acting on the rocket are gravity and engine thrust. The state variables for the model are the rocket's x and y positions, and the rocket's orientation angle α.

Thrust is a constant T_1 in the first stage, and steered using the programmed thrust angle θ. Thrust is resolved into x and y components by

$$T_{1x} = T_1 \cdot \cos(\alpha + \theta)$$

$$T_{1y} = T_1 \cdot \sin(\alpha + \theta).$$

The torque on the rocket is calculated by

$\Gamma = -T_1 \cdot \sin(\theta) \cdot L/2$, where L is the length of the rocket.

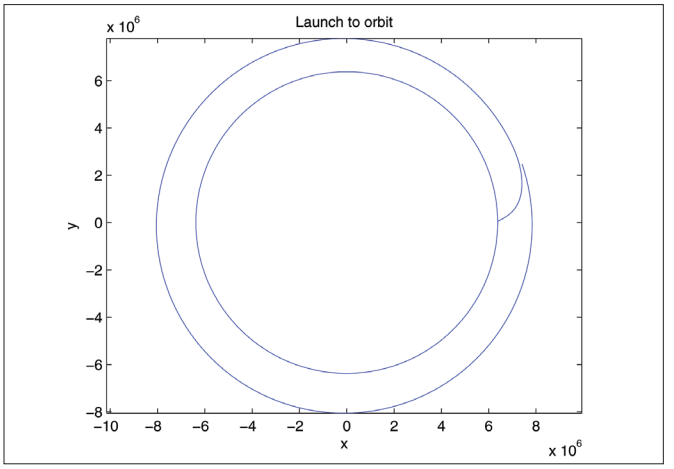


Fig. A5. IV rocket and model.

Figure A5 shows the rocket launched from the equator of a spinning Earth, inheriting the launch point's lateral velocity and the Earth's rotation rate, and steered into orbit using the programmed thrust angle θ. See the complete program at Ref 4. The program contains the bookkeeping necessary to model all three stages of the rocket's flight.

Figure A6 shows a VEX robot, frequently used in high school robotics courses and competitions.

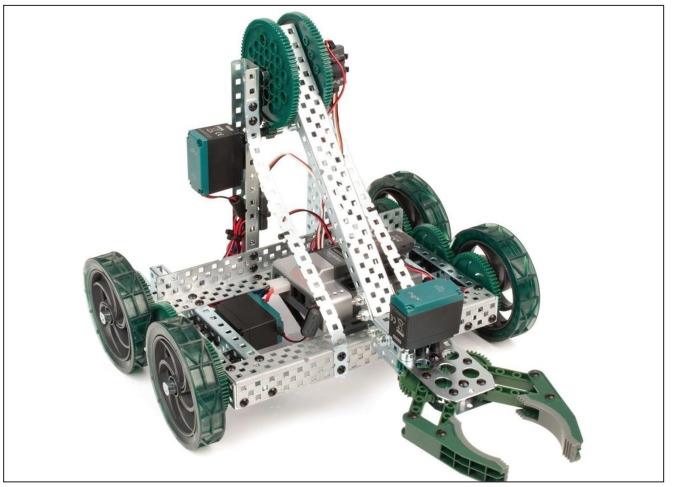


Fig. A6. VEX robot.

Figure A7 shows the circuit model for the robot drive motor. The state variables for the robot model are the circuit current I and the axle rotation rate ω.

An onboard computer controls the input voltage v_a. Current I through the circuit produces a torque $T_m = \mathrm{Km} \cdot I$ on the rotor. The rotor rotating at rate ω produces a back voltage in the circuit equal to $\mathrm{Kb} \cdot \omega$.

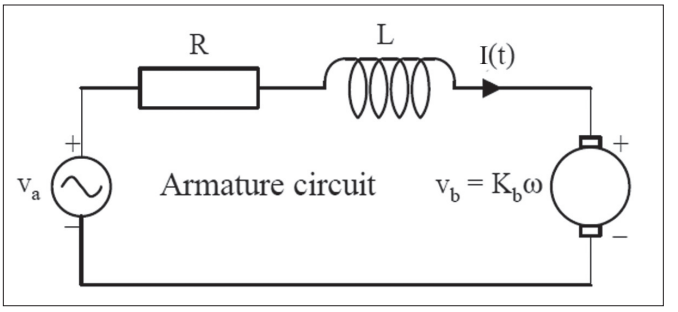


Fig. AA7. VEX robot drive circuit.

From Kirchoff's loop law and the resistor and inductor models

$$v_a(t) = R \cdot I(t) + L \cdot I'(t) + K_b \cdot \omega \,. \qquad (4)$$

Solving for $I'(t)$ gives

$$I'(t) = [v_a(t) - R \cdot I(t) - K_b \cdot \omega\,] / L. \qquad (5)$$

Figure A8 represents the robot armature/axle/wheel.

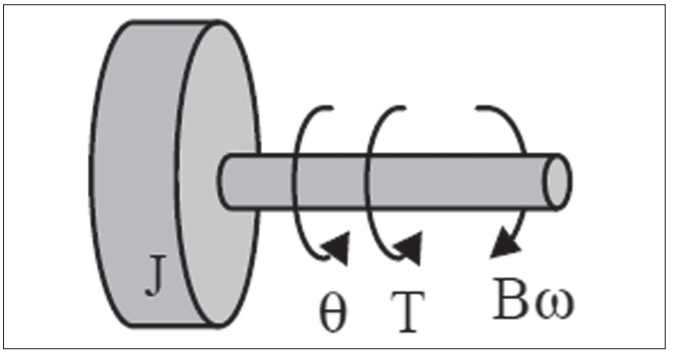


Fig. A8. Robot armature/axle/wheel model.

From Newton's second law for rotation

$$\omega' = (T - B \cdot \omega) / J, \tag{6}$$

where $T = T_m + T_f$ and J is the armature/axle/wheel moment of inertia. When the robot (mass m) accelerates the applied force is $m \cdot r \cdot \omega'$ (r = wheel radius). The floor exerts that force on the wheel and the applied torque is $T_f = m \cdot r^2 \cdot \omega'$.

Substituting for T in Eq. (6) and solving for ω' gives

$$\omega' = (T_m - B \cdot \omega) / (J + m \cdot r^2). \tag{7}$$

To keep track of the robot's position we add θ, the axle/wheel angle, as a state variable,

$$\theta' = \omega. \tag{8}$$

The for-loop code for the VEX robot model is:

```
for i=1:N
   t(i+1)=t(i)+dt;
   I(i+1) = I(i) + dt*(va(i)-R*I(i)- Kb*omega(i))/L;
   omegaprime = (Km*I(i)-B*omega(i))/(J + m*r^2);
   omega(i+1) = omega(i) + dt*omegaprime;
   theta(i+1) = theta(i) + dt*omega(i);
end          % See complete program at Ref. 4
```

Figure A9 is a graph of the system variables when v_a is a programmed 1-second unit pulse at time $t = 2$.

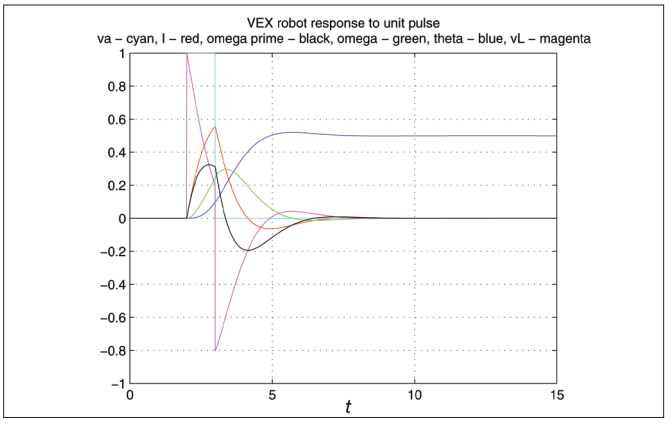


Fig. A9. Robot response to 1-second unit pulse.

Energy enters the system through v_a, is stored in the inductor magnetic field and the robot linear momentum and armature/axle/wheel angular momentum, and is lost through the resistor (R) and the rolling friction (B). If R and B are 0 the system forms an electro-mechanical oscillator, as shown in Fig. A10.

The state of the art

Simulation using modeling with differential equations and computational calculus is the state of the art for the analysis of most complex physical systems.

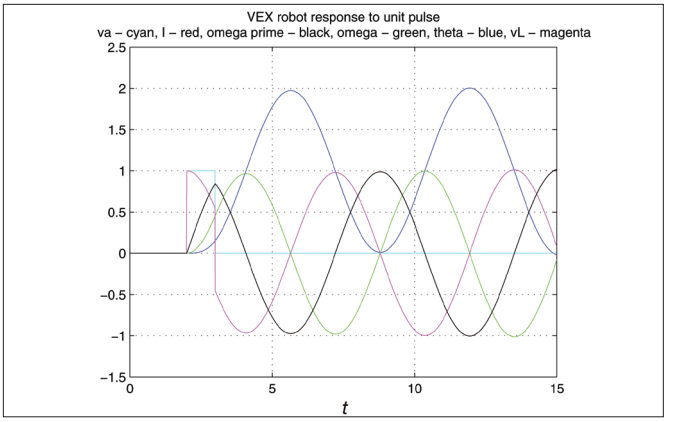


Fig. A10. Robot response to 1-second unit pulse when $R = B = 0$.

Central force motion—the Apollo and Juno trajectories are solutions to the three-body problem, which is analytically intractable, and hence solutions can only be computed numerically. Of course the simulations used in research and industry are 3D and more detailed than the ones in this paper.

Electric circuit analysis—SPICE (Simulation Program with Integrated Circuit Emphasis) is the industry standard tool used for circuit analysis.[13] The circuit designer enters a circuit description and the test parameters, and SPICE does the rest. SPICE is available online, easy to use, and can be demonstrated in class.

Models of 3D rigid body motion use Euler angles or quaternions to represent orientation and a tensor to represent the moment of inertia. We avoided these complications by restricting ourselves to 2D; as a result the models in this paper are simpler than those for realistic 3D problems. Once the model is derived, simulation using computational calculus is the state of the art method for analyzing 3D rigid body motion.

The state of the art algorithm for computing solutions to ordinary differential equations is the fourth-order Runge-Kutta algorithm.[3] Euler's method is the first-order Runge-Kutta algorithm. The higher level Runge-Kutta algorithms are based on Euler's method; they require fewer calculations to obtain more accurate results. The higher level Runge-Kutta methods are not difficult and can be presented in a single lecture; they are however, more tedious to program.

Partial differential equations

We have used Euler's method to analyze models written as ordinary differential equations. However, many branches of classical physics are based on partial differential equations, e.g. heat transfer, wave phenomena, stress and strain in materials, fluid dynamics, and electrodynamics. Euler's method extended to partial differential equations is known as the finite difference method. A course using the finite difference method to analyze physical systems based on partial differential equations is the logical follow on to the present course.[12]